Interface

Definition:

An interface describes actions. These actions can be implemented by classes.

In the Java programming language, an *interface* is a reference type, similar to a class, that can contain *only* constants, method signatures, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods. Interfaces cannot be instantiated—they can only be *implemented* by classes or *extended* by other interfaces.

The above definition is how java language uses interface concepts. But you need to know what the *interface* is in the real world.

Then only you have got an idea how to use *interfaces* in programming language. See the link below and read the green tick solution.

https://stackoverflow.com/questions/2866987/what-is-the-definition-of-interface-in-object-oriented-programming.

Rules for creating interfaces::

- Define constants (final variables)
- Abstracts methods
- Default methods
- Static methods
- Nested types

- we can't instantiate interfaces directly
- an interface can be empty, with no methods or variables in it (marker interface)
- we can't use the *final* word in the interface definition, as it will result in a compiler error
- all interface declarations should have the *public* or default access modifier; the *abstract* modifier will be added automatically by the compiler
- an interface method can't be *private*, *protected*, or *final*
- interface variables are *public*, *static*, and *final* by definition; we're not allowed to change their visibility.

Example of interface : interface always describes actions. It won't care about how to implement the actions.

Don't confuse class and interfaces.

Interface cares about actions. Classes care about how to build or develop these actions.

What is class:

Class is a blueprint which can be used to create an individual object. The blueprint can describe how to develop or implement actions.

What is interface:

Interface is a collection of actions. Later these actions are archived by classes.

Example :

Mobile has generally two actions which are 'call' and 'message'.

When mobile companies develop actual mobile they need to use these two actions(call and mobile). for example samsung company developing mobile devices.

- They have a blueprint of mobile which describes how to implement action like call and message.
- Important note: a blueprint is a detailed plan of action(call, message).

Consider example of `call`:

As of now call is *action* but we don't know how to make a call and what are the things to use to make a call.

When the Samsung company having a blueprint of mobile means they develop functionality of this action by using dialer and need to dial a certain number. Each mobile number is unique and it should be 10 digits.

In programmatically we can write below

```
public interface Mobile {
        abstract  void makeCall();
        abstract void doMessage();
}
```

By using classes we can implementing these actions

// blueprint which is used to create individual objects.

SamsungMobile class can be used to create multiple objects (samsung mobiles)

```
public  class SamsungMobile  implements Mobile {
        // override
        public void makeCall(){
                System.out. println(“make a call “);
        }
        public  void doMessage() {
                System.out. println(“send a message”);
        }
}
```

As of now we have calls of samsung mobile new we can start manufacturing mobiles

In programmatically we can create objects of SamsungMobile class.

```
SamsungMobile model1 = new SamsungMobile();
samsungMobile model2 = new SamsungMobile();
```

This is the journey of

interface => class => object

Mobile => SamsungMobile => model1

This is a basic interface, once you understand properly we will go further java 8 interface feature.